

# Sistemas Operativos 2018-19

## Guião da 1ª aula prática

LEIC-A / LEIC-T / LETI  
IST

Os alunos devem **ler primeiro o documento de visão geral do projeto** antes de lerem este guião. Este documento pretende guiar os alunos a realizar o exercício preparatório para o projeto da disciplina de Sistemas Operativos.

Este guia sugere alguns exercícios para os alunos se familiarizarem e completarem o código fornecido com uma implementação sequencial do algoritmo de Lee, denominada **CircuitRouter-SeqSolver**. Este programa será útil para o desenvolvimento do projeto e dos exercícios seguintes. Os exercícios permitem ainda praticar o ciclo de desenvolvimento de aplicações em linguagem C no ambiente UNIX.

Este exercício **não será avaliado**.

## 1 Contacto com o ambiente UNIX

1. Crie um diretório no seu computador e descarregue o arquivo `circuitrouter_seqsolver_ex01.zip` que está disponível na página da disciplina (no fénix), na secção “Laboratórios”.

Para extrair os ficheiros contidos no arquivo, use o comando

```
unzip circuitrouter_seqsolver_ex01.zip
```

2. Relembre o que fazem os comandos básicos como, por exemplo, `cd`, `ls`, `cat`, `cp`, `mv`, `rm`, `mkdir` e `rmdir`.

Recorde também que a generalidade dos comandos aceitam *switches* (também chamados *argumentos*, *opções* ou *flags*) que modificam o seu comportamento. Compare, por exemplo, o comportamento do comando `ls`, sem argumentos, com o comando `ls -l`.

Na secção seguinte detalha-se como pode obter ajuda ou informações sobre um certo comando em ambientes UNIX.

## 2 Utilização do manual

1. Pode aceder a informação detalhada sobre comandos de sistema, programas e funções da linguagem C, usando o comando `man` (abreviatura de “manual”), sob a forma das chamadas *manpages*.

Por exemplo, para se informar sobre o uso do próprio comando `man` deve escrever:

```
man man
```

Para navegar nas páginas do manual podem ser usadas as setas do teclado e as teclas “PageUp” e “PageDown”. Para sair do manual basta pressionar a tecla `q`.

2. O manual encontra-se organizado em secções numeradas de 1 a 9. Para a cadeira de Sistemas Operativos, as secções mais relevantes são:

- Secção 1: comandos/utilidades da *shell*
- Secção 2: chamadas de sistema
- Secção 3: funções de bibliotecas (e.g a biblioteca do C)

Isto é relevante pois existem comandos/funções com o mesmo nome que têm propósito e funcionamento diferentes.

Por exemplo, isso observa-se para o comando `printf` que está na **secção 1** e a função `printf` da linguagem C que está na **secção 3**. Ao invocar o manual, pode especificar a que secção pretende aceder, indicando o seu número antes do nome. Experimente os seguintes comandos:

```
man printf
```

```
man 3 printf
```

3. O manual também contém informação sobre programas/ferramentas. Por exemplo, para consultar a *manpage* do comando `zip`:

```
man zip
```

Outra forma de obter informação recorre directamente aos programas/ferramentas e ao uso do *switch* `--help`, que é geralmente suportado:

```
zip --help
```

Adicionalmente, pode experimentar o *switch* `--help` ou consultar a *manpage*, por exemplo, dos comandos `gdb`, `gcc`, `make`.

4. O uso do manual é especialmente útil para obter informação sobre as funções do C e identificar os valores devolvidos – notar a secção **RETURN VALUE**. Este aspeto é muito importante, pois nenhum programa deve chamar uma função e, no retorno, ignorar se ocorreu alguma situação de erro durante a execução da função. Como regra, antes de usar uma função, os alunos devem estudar nas *man pages* as diversas situações de erro que podem ocorrer e assegurar que o programa as trata devidamente (analisando o retorno da função).

### 3 Introdução à *shell programming*

1. Em ambientes UNIX existem três conceitos importantes: `stdin`, `stdout` e `stderr`.
  - O `stdin` (“standard input”) representa o dispositivo de entrada de um programa – tipicamente o teclado;
  - O `stdout` (“standard output”) representa o dispositivo de saída – tipicamente o terminal;
  - O `stderr` (“standard error”) representa um dispositivo alternativo de saída para mensagens de erro, que, por defeito, é o mesmo dispositivo que o `stdout`.
2. É possível redirecionar qualquer um destes dispositivos para ficheiros usando *redirection operators* (`<`, `>`, `&>`, `>>`, ...). Experimente executar os seguintes comandos, examinando o conteúdo da diretoria atual, e dos ficheiros referidos, após cada um deles:

```
echo Hello World
```

```
echo Hello World > my_stdout.txt
```

```
echo Hello again >> my_stdout.txt
```

```
echo Goodbye > my_stdout.txt
```

```
cat my_stdout.txt nonexistent_file 2> my_stderr.txt
```

```
cat my_stdout.txt nonexistent_file &> my_stdout_and_stderr.txt
```

```
cat < my_stdout.txt
```

**NOTA:** as sintaxes apresentadas em cima representam apenas alguns exemplos. Durante as aulas teóricas serão descritas formas mais genéricas de redirecionar os canais de Entrada/Saída dos processos Unix. No entanto, podem encontrar já detalhes sobre os *redirection operators* na secção REDIRECTION da *manpage* do *bash* (`man bash`) ou em <https://www.tldp.org/LDP/abs/html/io-redirection.html>.

3. É também possível redirecionar o `stdout` de um comando para o `stdin` de outro, criando assim uma cadeia de comandos para processar informação. Por exemplo, a seguinte cadeia de comandos lê o conteúdo do ficheiro `/etc/passwd`, filtra as linhas que contenham a palavra `root` e imprime a 7ª coluna (separadas pelo carácter `:`) de cada linha:

```
cat /etc/passwd | grep root | cut -d : -f 7
```

Estas redireções são feitas com recurso a *pipes*, conceito que será abordado mais a fundo durante as aulas teóricas.

## 4 Análise do programa CircuitRouter-SeqSolver fornecido

1. Analise os ficheiros extraídos do arquivo `circuitrouter_seqsolver_ex01.zip` usando o editor de texto da sua preferência (e.g. `vim`, `emacs`, `nano`, `gedit`).

O problema de *routing* de circuitos pode ser reduzido ao problema de encontrar o caminho mais curto entre dois pontos numa grelha tridimensional. O algoritmo de *routing* de Lee, apresentado no documento de visão geral do projeto, resolve este último problema. Eis um resumo das principais estruturas de dados utilizadas:

**Grid** A estrutura de dados `grid_t` é usada para representar uma grelha. Esta estrutura guarda o número de linhas, colunas, e camadas (uma vez que o circuito pode ser tridimensional), e um apontador para um vector de variáveis do tipo `long`, onde são guardados os valores de cada ponto da grelha.

**Maze** A estrutura de dados `maze_t` é utilizada para guardar as variáveis do problema, tais como a grelha, os componentes do circuito e obstáculos.

**Router** Por fim, a estrutura `router_solve_arg` contém alguns dados de entrada, assim como o resultado da aplicação do algoritmo de *routing* de Lee, nomeadamente os custos para cada direcção (`router_t`), o *maze*, e uma lista com os caminhos encontrados.

O código fornecido inclui diversas funções que manipulam estas estruturas. As funções mais importantes para a aplicação do algoritmo estão definidas no ficheiro `router.c`. Analise cuidadosamente as funções `doExpansion` e `doTraceback`, que realizam as funções de expansão e retrocesso do algoritmo.

Note que a implementação do algoritmo inicia a contagem em 0, e não em 1, tal como ilustrado no exemplo do enunciado geral.

2. Observe a seguinte grelha:

					$d_1$		
	$d_2$						
	$s_1$						
		$s_2$					

Usando papel e lápis, simule a execução do algoritmo na grelha acima, mostrando o estado da grelha após a fase de expansão e após a fase de retrocesso. Repita o processo pela ordem inversa (i.e. se começou por expandir  $s_1$ , comece a expansão por  $s_2$ , e vice versa).

## 5 Geração e modificação do programa CircuitRouter-SeqSolver

1. Gere o programa `CircuitRouter-SeqSolver` e execute-o usando os seguintes comandos:

```

1 gcc -c coordinate.c -o coordinate.o
2 gcc -c grid.c -o grid.o
3 gcc -c CircuitRouter-SeqSolver.c -o CircuitRouter-SeqSolver.o
4 gcc -c maze.c -o maze.o
5 gcc -c router.c -o router.o
6 gcc -c lib/list.c -o lib/list.o
7 gcc -c lib/pair.c -o lib/pair.o
8 gcc -c lib/queue.c -o lib/queue.o
9 gcc -c lib/vector.c -o lib/vector.o
10 gcc coordinate.o grid.o CircuitRouter-SeqSolver.o maze.o router.o lib/list.o lib/pair.o lib/queue.o
    lib/vector.o -lm -o CircuitRouter-SeqSolver
11 ./CircuitRouter-SeqSolver < inputs/random-x32-y32-z3-n64.txt

```

**Atenção:** insiram os comandos manualmente (e analisem-nos atentamente!), pois ao fazer *Copy&Paste* deste documento para a *shell* poderão ser introduzidos erros.

2. Pela análise do ficheiro `maze.c`, na função `maze_read`, verifica-se que o programa recebe alguns dados de entrada através do `stdin`, nomeadamente:

- Dimensão da grelha (formato: `d <x> <y> <z>`)
- Caminhos (formato: `p <x1> <y1> <z1> <x2> <y2> <z2>`)
- Obstáculos (formato: `w <x> <y> <z>`)

Para além destes dados, aceita também alguns argumentos diretamente na linha de comandos. Para os listar e obter uma descrição dos mesmos, experimente executar o programa com a flag **-h**:

```
./CircuitRouter-SeqSolver -h
```

Verifique que validações são feitas aos dados de entrada e experimente correr o programa com várias combinações de inputs, incluindo com pontos fora da grelha (por exemplo, com coordenadas negativas).

3. Adicione validações para garantir que apenas são aceites parâmetros com coordenadas válidas e teste novamente os inputs utilizados na alínea anterior.

**Sugestão:** analise a função `addToGrid` do ficheiro `maze.c`.

4. Implemente uma função que imprima o estado atual da grelha no ecrã. Mostre cada camada da grelha tridimensional individualmente, indicando o valor de  $z$  da camada atual. Utilize a seguinte representação:

**0** Ponto de conexão

**-1** Ponto vazio

**-2** Obstáculo

$n \in \mathbb{N}$  Rota que liga o par  $n$

Repare que a grelha já é preenchida com estes valores ao longo da execução do algoritmo.

Exemplo de output para o circuito analisado durante a simulação em papel:

```
[z = 0]
-1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 0 -1 -1
-1 -1 -1 -1 -1 1 -1 -1
 2 0 -1 -1 -1 1 -1 -1
 2 0 1 1 1 1 -1 -1
 2 2 0 -1 -1 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1
```

Note que a função `grid_print` já se encontra declarada no ficheiro `grid.c`, sendo apenas necessário completá-la.

5. Utilize as funções desenvolvidas na alínea anterior para observar o estado da grelha no final da fase de expansão e depois da fase de “trace-back” usando o mesmo circuito analisado durante a simulação em papel da execução do algoritmo.